INF1010

Programmation Orientée-Objet

Travail pratique #5
Classes génériques et la STL

Objectifs :	Permettre à l'étudiant de se familiariser avec le concept de fonctions et des classes génériques, aux foncteurs, aux conteneurs et algorithmes de la STL
Remise du travail :	Lundi 9 Avril 2017, 8h
Références :	Notes de cours sur Moodle & Chapitres 11 à 14, 16 et 20 du livre Big C++ 2e éd.
Documents à remettre :	Les fichiers .cpp et .h complétés, les questions en pdf le tout sous la forme d'une archive au format .zip.
Directives :	Directives de remise des Travaux pratiques sur Moodle Les travaux dirigés s'effectuent obligatoirement en équipe de deux personnes faisant partie du même groupe. Veuillez suivre le guide de codage

Informations préalables

Veuillez lire attentivement tout l'énoncé avant de commencer à écrire du code, afin de vous assurer de bien comprendre l'interaction entre les classes. Cela vous permettra aussi de commencer par ce qui vous semble le plus pertinent, car en effet l'ordre de description des classes dans le TP n'est pas forcément l'ordre à suivre. Il pourrait par exemple être plus intéressant de faire toutes les méthodes liées à l'insertion d'une donnée pour pouvoir les tester et continuer par la suite.

Notez que ce TP est sans doute le plus difficile de la session, c'est pour cela que vous avez 3 séances pour le réaliser, attention à ne pas prendre de retard et de bien répartir la charge de travail jusqu'au 9 Avril. Assurez-vous aussi de bien comprendre le fonctionnement des templates et leur utilité pour créer des classes génériques. En complément au cours et la documentation de la STL, vous pouvez consulter ce lien.

Le travail effectué dans ce TP continue celui amorcé par les TP 1, 2, 3 et 4, soit une simulation de e-commerce en y intégrant les notions de foncteurs, classes génériques, les conteneurs et les algorithmes de la STL.

Les foncteurs sont des objets qui peuvent agir comme des fonctions grâce à la surcharge de l'opérateur « () ». Les foncteurs sont très utiles lorsqu'on travaille avec des structures de données (conteneurs) provenant de la bibliothèque STL comme une list ou une map... Les foncteurs peuvent avoir aucun ou plusieurs attributs. Généralement, un foncteur sans attribut permet de manipuler les objets sur lesquels il est appliqué. Les foncteurs à un attribut sont souvent utilisés à des fins de comparaison, mais aussi d'autres utilités.

L'utilisation du **static_cast** est **INTERDITE**. Tout le TP est réalisable sans cette fonction.

Pour vous aider, les fichiers du TP précédent vous sont fournis. Vous n'avez qu'à implémenter les nouvelles méthodes décrites plus bas. Les attributs ou méthodes qui ne sont plus nécessaires, ont été supprimés. Et les méthodes à modifier vous ont été indiquées.

ATTENTION : Tout au long du TP, assurez-vous d'utiliser les opérateurs sur les objets et non sur leurs pointeurs ! Vous devez donc déréférencer les pointeurs si nécessaires.

ATTENTION : Vous serez pénalisés pour les utilisations inutiles du mot-clé *this*. Utilisez-le seulement où nécessaire.

ATTENTION : Il est fortement recommandé d'utiliser les fichiers fournis, plutôt que de continuer avec vos fichiers du TP4.

ATTENTION : L'utilisation de boucles for ou while de la forme for (int i; i <vec.size(); i++) est interdit pour les nouvelles méthodes que vous devez implémenter. Vous devez soit utiliser les algorithmes lorsque possible, soit utiliser les boucles for/while en utilisant les itérateurs.

Remarque : Pour plus de précision sur le travail à faire et les changements à effectuer, veuillez-vous référer aux fichiers .h.

Travail à réaliser

Lisez la description des classes ci-dessous et suivez aussi les indications des **TODO** insérées dans le code pour apporter vos modifications au code fourni.

Si une méthode d'affichage ne respecte le format de la capture donnée en annexe, vous devez modifier les méthodes en question.

Classes Produit, ProduitSolde, ProduitAuxEncheres

Les trois classes qui servent à caractériser un produit, restent inchangées par rapport au TP4. La classe Produit a intégré la classe ProduitOrdinaire.

Classe *Usager*

La classe Usager reste inchangé par rapport au TP4.

Classe Fournisseur

Cette classe caractérise un fournisseur de produits et comme pour le précédent TP elle dérive d'Usager.

Attributs :

- **catalogue_ :** était un vecteur de Produit* qui a été remplacé par un pointeur vers un GestionnaireProduits devant être instancié dans les constructeurs.

Méthodes à ajouter :

- **trouverProduitPlusCher() :** retourne le produit le plus cher du catalogue en appelant la méthode adéquate.
- **diminuerPrix() :** prend en paramètre un entier (pourcentage de diminution) et diminue le prix de tous les produits du catalogue_ en appelant la méthode de GestionnaireProduits avec le prédicat adéquat.

Méthodes à modifier :

- Les autres méthodes à modifier doivent simplement appeler les bonnes méthodes du catalogue_.

Classe *Client*

La classe Client dérive de la classe de base Usager.

Remarque : Les modifications doivent utiliser les méthodes de la classe set de la STL ainsi que les itérateurs de set. Vous avez le droit d'utiliser le mot clé « auto » pour simplifier votre code.

Attributs :

- **panier_ :** était un vecteur de Produit* qui a été remplacé par un pointeur vers un GestionnaireProduits devant être instancié dans les constructeurs.

Méthode à ajouter :

- **trouverProduitPlusCher() :** retourne le produit le plus cher du catalogue en appelant la méthode adéquate.

Méthodes à modifier :

- Les autres méthodes à modifier doivent simplement appeler les bonnes méthodes du panier.

Classe *GestionnaireGenerique*

Cette classe permet la création d'un gestionnaire générique qui permet de manipuler plusieurs données d'un même **type T** (dans notre cas des usagers ou des produits) à l'aide d'un conteneur de **type C** (qui pourra par exemple soit être une multimap ou un set). Comme les méthodes d'ajout et de suppression sont différentes pour chaque conteneur nous allons utiliser des foncteurs d'ajout de **type A** et de suppression de **type S** propre au type de conteneur utilisé.

Attributs :

- (protected) **conteneur_** : Le conteneur qui contiendra les éléments du gestionnaire.

Méthodes :

- **obtenirConteneur() :** retourne le conteneur du gestionnaire.
- **ajouter() :** prend en paramètre un type T* qui devra ajouter ce pointeur dans le conteneur à l'aide d'un foncteur d'ajout qui changera en fonction du type du conteneur. Cette méthode n'a pas de valeur de retour.
- **supprimer() :** prend en paramètre un type T* qui devra supprimer ce pointeur dans le conteneur à l'aide d'un foncteur de suppression en fonction du type de conteneur. Cette méthode n'a pas de valeur de retour.
- **pourChaqueElement() :** est un méthode générique de template<typename **Predicate**> qui prend en paramètre un foncteur qui agira comme un prédicat. Ce prédicat sera appliqué à tous les éléments du conteneur à l'aide d'un **for_each**.

Classe GestionnaireUsager

Cette classe hérite de GestionnaireGenerique pour permettre la manipulation d'usagers. Dans notre commerce, on veut qu'un usager soit unique et identifié par sa référence, on utilisera donc un set de type **set<Usager*>** avec les foncteurs d'ajout et de suppression appropriés.

La classe ne contient pas d'attribut seulement des méthodes qui sont les mêmes que pour le TP4 mais qui devront **être modifiées pour fonctionner avec le nouveau code.**

Classe GestionnaireProduit

Cette classe hérite de GestionnaireGenerique pour permettre la manipulation de produit. Dans notre commerce, un usager peut posséder plusieurs produits du même type, on utilisera donc une multimap de type **multimap <int, Produit*>** avec les foncteurs d'ajout et de suppression appropriés.

La classe ne contient pas d'attributs seulement des méthodes qui sont les mêmes que pour le TP4 mais qui devront **être modifiées pour fonctionner avec le nouveau code.**

Méthodes :

- **reinitialiserClient() :** réinitialise la multimap de Produit, comment on le faisait pour la méthode reinitialiser() de la classe Client dans le TP4.
- **reinitialiserFournisseur() :** Réinitialise la multimap de Produit, comment on le faisait pour la méthode reinitialiser() le la classe Client dans le TP4.
- **afficher() :** affiche les produits de la multimap.
- **obtenirTotalAPayer() :** Calcul le total à payer du Client, comme on le faisait dans la méthode obtenirTotalAPayer() de la Classe Client dans le TP4.
- **obtenirTotalApayerPremium() :** Calcul le total à payer du ClientPremium, comme on le faisait dans la méthode obtenirTotalAPayer() de la Classe Client dans le TP4.
- **trouverProduitPlusCher()** : retourne le Produit ayant le prix le plus élevé dans la multimap de produits. Pour ce faire vous vous servirez de la fonction max_element de la STL à l'intérieur de laquelle vous utiliserez une fonction lambda pour comparer deux éléments de la multimap.
- **obtenirProduitsEntre()** : prend deux paramètres de type double qui représenteront les bornes de l'intervalle. Elle doit utiliser le FoncteurIntervalle dans une fonction **copy_if** de la STL en utilisant un **back_inserter**. Cette méthode retourne un vector contenant les pairs **pair<int, Produit*>** dont les prix des Produits associés sont compris dans cet intervalle.
- **obtenirProduitSuivant()** : prend en paramètre un pointeur de Produits et retourne le pointeur de Produit ayant une référence strictement supérieur. Pour cela, vous devez utiliser la fonction **find_if** de la STL, à l'intérieur duquel vous utiliserez la fonction **bind** de la STL.

Ce fichier comporte les différents foncteurs utilisés dans le TP. Les foncteurs doivent tous être implémentés uniquement dans ce fichier.

FoncteurEgal

C'est un foncteur générique qui compare deux objets.

Attributs :

- **t_ :** de type T* initialisé par paramètre dans son constructeur.

Méthodes :

- **operator() :** prend un paramètre une **pair<int, T*>** et retourne vrai si la valeur T* de la pair est égale à son attribut.

FoncteurGenerateurId

C'est un foncteur qui ne prend pas d'argument et qui permet de générer des ID unique. Dans notre cas, il sera utilisé dans le main. Un premier permettra de générer les ID pour les clients et le second pour les produits. Pour faire la génération unique, le foncteur initialisera son attribut **id_** à 0 et s'incrémentera de 1 à chaque appel de celui-ci.

Attributs :

- **id_ :** est un entier initialisé à 0 (zéro) dans le constructeur.

Méthodes :

- **operator() :** ne prend pas de paramètre et incrémente l'attribut de 1.

FoncteurDiminuerPourcent

Ce foncteur est utilisé par le fournisseur et servira à diminuer d'un certain pourcentage le prix d'un produit.

Attributs :

- **pourcentage_ :** est un entier initialisé par paramètre dans le constructeur.

Méthodes :

- **operator() :** prend en paramètre une **pair<int, Produit*>** et doit réduire de pourcentage_ le prix du produit associé. Faire attention à ne pas appliquer la réduction sur le prix soldé, mais unique sur le prix brut (sans dynamic_cast).

FoncteurIntervalle

Ce foncteur est utilisé par le GestionnaireProduits afin de trouver le Produit de la **multimap** dont le prix est compris entre borneInf_ et borneSup_.

Attributs :

- **borneInf_ :** est un double initialisé par paramètre dans le constructeur.
- **borneSup_ :** est un double initialisé par paramètre dans le constructeur.

Méthodes :

operator() : prend en paramètre un **pair<int, Produit*>** et retourne vrai si le prix du Produit associé à la pair est compris entre les bornes borneInf_ et borneSup_, faux sinon.

AjouterProduit

Ce foncteur permet l'ajout d'un produit dans un conteneur de type **multimap**.

Attribut :

- **&multimap_ :** de type **multimap<int, Produit*>** est une agrégation par référence et initialisé par paramètre (par référence) dans le constructeur.

Méthodes :

- **operator() :** prend en paramètre un pointeur de Produit et retourne la multimap par référence avec le produit ajouté.

SupprimerProduit

Ce foncteur permet la suppression d'un produit dans un conteneur de type **multimap**.

Attribut :

- **&multimap_ :** de type **multimap<int, Produit*>** est une agrégation par référence et est initialisé par paramètre (par référence) dans le constructeur.

Méthode :

- **operator() :** prend en paramètre un pointeur de Produit, utilise la fonction **find_if** de la STL avec le FoncteurEgal comme prédicat, vérifie si le Produit existe et retourne la multimap par référence avec le produit supprimé.

AjouterUsager

Ce foncteur permet l'ajout d'un usager dans un conteneur de type **set**.

Attribut :

- **&set_ :** de type **set<Usager*>** est initialisé par paramètre (par référence) dans le constructeur.

Méthode :

- **operator() :** prend en paramètre un pointeur d'Usager et retourne la set par référence avec le produit ajouté.

SupprimerUsager

Ce foncteur permet la suppression d'un usager dans un conteneur de type set.

Attribut :

- **&set_ :** de type **set<Usager*>** est initialisé par paramètre (par référence) dans le constructeur.

Méthode :

- **operator() :** prend en paramètre un pointeur d'Usager, vérifie que l'Usager existe bien, le supprime et retourne la set par référence avec le produit supprimé.

Main.cpp

Pour ce TP le main est entièrement fournis avec des tests comme pour les TPs précédents, n'apportez **aucune modification à celui-ci** et répondez aux questions cette fois ci dans un document PDF que vous ajouterez à l'archive de votre remise.

Si certains tests ne fonctionnent pas, regardez attentivement dans le main celui qui ne passe pas et ajustez votre code en conséquence. Pendant la correction, nous remplacerons votre main par le nôtre afin de prévenir toutes modifications de votre part.

Spécifications générales

• Ajouter un destructeur pour chaque classe chaque fois que cela vous semble pertinent.
• Utilisez la liste d'initialisation pour l'implémentation de vos constructeurs
• Ajouter le mot-clé const chaque fois que cela est pertinent
• Appliquez un affichage similaire à ce qui est présenté à la fin de ce document.
• Documenter votre code source.

• Répondez aux questions dans un PDF que vous joindrez à l'archive zip de votre remise. **Attention, les réponses rédigées dans le main ne seront pas corrigées.**

Questions

1. Est-ce qu'on aurait pu utiliser une map au lieu d'un set pour notre gestionnaire d'usagers, pourquoi ? Si oui, donnez l'initialisation de la map qui remplacerait le set du GestionnaireUsagers?

2. Pourquoi est-ce que l'implémentation des classes génériques est dans .h et non pas séparée en .h et .cpp comme les classes normales ?

1. Donner le diagramme des classes dans le fichier PDF.

La correction du TP se fera sur 20 points.

Voici les détails de la correction :

- (4 points) Compilation du programme ;
- (3 points) Exécution du programme ;
- (5 points) Comportement exact des méthodes du programme et l'utilisation de la STL;
- (4 points) Utilisation adéquate des foncteurs;
- (1 points) Documentation du code ;
- (2 points) Allocation / désallocation appropriée mémoire
- (1 point) Réponse aux questions.

Affichage attendu

```
PANIER (de Donada--Vidal)
        chaussures
                reference:      1
                prix:           $36
                exemplaires :   1
        montre
                reference:      6
                prix:           $90
                rabais:         30%
                exemplaires :   1

PANIER (de Cash)
        chaussures
                reference:      1
                prix:           $36
                exemplaires :   1
        violoncelle
                reference:      4
                prix:           $21000
                prix initial:   5000
                encherisseur:   Cash
                exemplaires :   1
        montre
                reference:      6
                prix:           $90
                rabais:         30%
                exemplaires :   1
        nem crevettes
                reference:      7
                prix:           $2
                rabais:         0%
                exemplaires :   1

PANIER (de Doe)

CATALOGUE (de Bellaiche)

CATALOGUE (de Kadoury)
        chaussures
                reference:      1
                prix:           $36
                exemplaires :   1
        montre
                reference:      6
                prix:           $90
                rabais:         30%
                exemplaires :   1
```

```
CATALOGUE (de Doe)

PROFILS
        Kadoury, Samuel (7)
                code postal:    H1G 2G4
                catalogue:      2 elements
        Bellaiche, Martine (6)
                code postal:    H4C 8D4
                catalogue:      0 elements
        Doe, John (5)
                code postal:    A1A A1A
                catalogue:      0 elements
        Cash, Julie (4)
                code postal:    HZ9 1J4
                code client:    19141918
                panier:         4 elements
                jours restants: 50
        Doe, John (3)
                code postal:    A1A A1A
                code client:    0
                panier:         0 elements
                jours restants: 0
        S, Rick (2)
                code postal:    HF1 8H3
                code client:    20012003
                panier:         0 elements
        Donada--Vidal, Gaspard (1)
                code postal:    P4R 1I5
                code client:    1997
                panier:         2 elements
        Doe, John (0)
                code postal:    A1A A1A
                code client:    0
                panier:         0 elements
```

```
TESTS
        Test 01... OK!
        Test 02... OK!
        Test 03... OK!
        Test 04... OK!
        Test 05... OK!
        Test 06... OK!
        Test 07... OK!
        Test 08... OK!
        Test 09... OK!
        Test 10... OK!
        Test 11... OK!
        Test 12... OK!
        Test 13... OK!
        Test 14... OK!
        Test 15... OK!
        Test 16... OK!
        Test 17... OK!
        Test 18... OK!
        Test 19... OK!
        Test 20... OK!
        Test 21... OK!
        Test 22... OK!
        Test 23... OK!
        Test 24... OK!
        Test 25... OK!
        Test 26... OK!
        Test 27... OK!
        Test 28... OK!
        Test 29... OK!
        Test 30... OK!
        Test 31... OK!
        Test 32... OK!
        Test 33... OK!
        Test 34... OK!
        Test 35... OK!
        Test 36... OK!
        Test 37... OK!
        Test 38... OK!
        Test 39... OK!
        Test 40... OK!
        Test 41... OK!
        Test 42... OK!
        Test 43... OK!
        Test 44... OK!
        Test 45... OK!
        Test 46... OK!
        Test 47... OK!
        Test 48... OK!
        Test 49... OK!
        Test 50... OK!
        Test 51... OK!
        Test 52... OK!
```